Understanding and Mitigating the Limitations of Prioritized Experience Replay (Supplementary material)

Yangchen Pan[1,3] **Jincheng Mei**[*1] **Amir-massoud Farahmand**[2,4] **Martha White**[1,4]
Hengshuai Yao[1] **Mohsen Rohani**[3] **Jun Luo**[3]

[1]University of Alberta
[2]University of Toronto & Vector Institute
[3]Huawei Noah's Ark Lab
[4]CIFAR AI Chair

1 APPENDIX

The appendix includes the following contents:

1. Section 1.1: background in Dyna architecture and two Dyna variants.
2. Section 1.2: background of Langevin dynamics.
3. Section 1.3: the full proof of Theorem 1.
4. Section 1.4: the full proof of Theorem 2 and its simulations.
5. Section 1.5: the theorem characterizing the error bound between the sampling distribution estimated by using a true model and a learned model. It includes the full proof.
6. Section 1.6: a discussion and some empirical study of using high power objectives.
7. Section 1.7: supplementary experimental results: training error results to check the negative effects of the limitations of prioritized sampling; results to verify the equivalence between prioritized sampling and cubic power; additional results on the discrete control domains 1.7.3; results on the autonomous driving application; results on MazeGridWorld from Pan et al. [2020].
8. Section 1.8: details for reproducible research.

1.1 BACKGROUND IN DYNA

Dyna integrates model-free and model-based policy updates in an online RL setting [Sutton, 1990]. As shown in Algorithm 1, at each time step, a Dyna agent uses the real experience to learn a model and performs a model-free policy update. During the *planning* stage, simulated experiences are acquired from the model to further improve the policy. It should be noted that, in Dyna, the concept of *planning* refers to any computational process which leverages a model to improve policy, according to Sutton and Barto [2018], Chapter 8. The mechanism of generating states or state-action pairs from which to query the model is called *search-control*, which is of critical importance to improving sample efficiency. The below algorithm shows a naive search-control strategy: simply use visited state-action pairs and store them into the search-control queue. During the planning stage, these pairs are uniformly sampled according to the original paper.

The recent works by Pan et al. [2019, 2020] propose two search-control strategies to generate states. The first one is to search high-value states actively, and the second one is to search states whose values are difficult to learn.

However, there are several limitations of the two previous works. First, they do not provide any theoretical justification to use the stochastic gradient ascent trajectories for search-control. Second, HC on gradient norm and Hessian norm of

[*]Work done while at the University of Alberta. Equal contribution with Yangchen Pan. Correspondence to: Yangchen Pan <pan6@ualberta.ca> and Jincheng Mei <jmei2@ualberta.ca>.

Accepted for the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022).

the learned value function [Pan et al., 2020] suffers from great computation cost and zero or explosive gradient due to the high order differentiation (i.e., $\nabla_s||\nabla_s v(s)||$) as suggested by the authors. When using ReLu as activation functions, such high order differentiation almost results in zero gradients. We empirically verified this phenomenon. And this phenomenon can also be verified by intuition from the work by Goodfellow et al. [2015], which suggests that ReLU neural networks are locally almost linear. Then it is not surprising to have zero higher order derivatives. Third, the two methods are prone to result in sub-optimal policies: consider that the values of states are relatively well-learned and fixed, then value-based search-control (Dyna-Value) would still find those high-value states even though they might already have low TD error.

Algorithm 1 Tabular Dyna

```
Initialize $Q(s,a)$; initialize model $\mathcal{M}(s,a), \forall(s,a) \in \mathcal{S} \times \mathcal{A}$
while true do
    observe $s$, take action $a$ by $\epsilon$-greedy w.r.t $Q(s,\cdot)$
    execute $a$, observe reward $R$ and next State $s'$
    Q-learning update for $Q(s,a)$
    update model $\mathcal{M}(s,a)$ (i.e. by counting)
    store $(s,a)$ into search-control queue // this is a naive search-control strategy
    for i=1:d do
        sample $(\tilde{s},\tilde{a})$ from search-control queue
        $(\tilde{s}',\tilde{R}) \leftarrow \mathcal{M}(\tilde{s},\tilde{a})$ // simulated transition
        Q-learning update for $Q(\tilde{s},\tilde{a})$ // planning updates/steps
```

1.2 DISCUSSION ON THE LANGEVIN DYNAMICS MONTE CARLO METHOD

Theoretical mechanism. Define a SDE: $\mathrm{d}W(t) = \nabla U(W_t)\mathrm{d}t + \sqrt{2}\mathrm{d}B_t$, where $B_t \in \mathbb{R}^d$ is a d-dimensional Brownian motion and U is a continuous differentiable function. It turns out that the Langevin diffusion $(W_t)_{t\geq 0}$ converges to a unique invariant distribution $p(x) \propto \exp(U(x))$ [Chiang et al., 1987]. By applying the Euler-Maruyama discretization scheme to the SDE, we acquire the discretized version $Y_{k+1} = Y_k + \alpha_{k+1}\nabla U(Y_k) + \sqrt{2\alpha_{k+1}}Z_{k+1}$ where $(Z_k)_{k\geq 1}$ is an i.i.d. sequence of standard d-dimensional Gaussian random vectors and $(\alpha_k)_{k\geq 1}$ is a sequence of step sizes. It has been proved that the limiting distribution of the sequence $(Y_k)_{k\geq 1}$ converges to the invariant distribution of the underlying SDE [Roberts, 1996, Durmus and Moulines, 2017]. As a result, considering $U(\cdot)$ as $\log|\delta(\cdot)|$, Y as s justifies our SGLD sampling method.

1.3 PROOF FOR THEOREM 1

Theorem 1. For a constant c determined by $\theta, \mathcal{T}$, we have

$$\mathop{\mathbb{E}}_{(x,y)\sim uniform(\mathcal{T})}\left[\frac{1}{3}\cdot\frac{\partial\,|f_\theta(x)-y|^3}{\partial\theta}\right] = c\cdot \mathop{\mathbb{E}}_{(x,y)\sim q(x,y;\theta)}\left[\frac{1}{2}\cdot\frac{\partial\,(f_\theta(x)-y)^2}{\partial\theta}\right]$$

Proof. For the l.h.s., we have,

$$\mathop{\mathbb{E}}_{(x,y)\sim uniform(\mathcal{T})}\left[\frac{1}{3}\cdot\frac{\partial\left|f_\theta(x)-y\right|^3}{\partial\theta}\right] \tag{1}$$
$$=\frac{1}{3\cdot n}\cdot\sum_{i=1}^{n}\frac{\partial\left|f_\theta(x(i))-y(i)\right|^3}{\partial\theta} \tag{2}$$
$$=\frac{1}{3\cdot n}\cdot\sum_{i=1}^{n}\frac{\partial\Big(\left(f_\theta(x(i))-y(i)\right)^2\Big)^{\frac{3}{2}}}{\partial\theta} \tag{3}$$
$$=\frac{1}{3\cdot n}\cdot\sum_{i=1}^{n}\frac{\partial\left(\left(f_\theta(x(i))-y(i)\right)^2\right)^{\frac{3}{2}}}{\partial\left(f_\theta(x)-y\right)^2}\cdot\frac{\partial\left(f_\theta(x(i))-y(i)\right)^2}{\partial\theta} \tag{4}$$
$$=\frac{1}{2\cdot n}\cdot\sum_{i=1}^{n}\left|f_\theta(x(i))-y(i)\right|\cdot\frac{\partial\left(f_\theta(x(i))-y(i)\right)^2}{\partial\theta}. \tag{5}$$

On the other hand, for the r.h.s., we have,

$$\mathop{\mathbb{E}}_{(x,y)\sim q(x,y;\theta)}\left[\frac{1}{2}\cdot\frac{\partial\left(f_\theta(x)-y\right)^2}{\partial\theta}\right] \tag{6}$$
$$=\frac{1}{2}\cdot\sum_{i=1}^{n}q(x_i,y_i;\theta)\cdot\frac{\partial\left(f_\theta(x(i))-y(i)\right)^2}{\partial\theta} \tag{7}$$
$$=\frac{n}{\sum_{j=1}^{n}\left|f_\theta(x_j)-y_j\right|}\cdot\left[\frac{1}{2\cdot n}\cdot\sum_{i=1}^{n}\left|f_\theta(x(i))-y(i)\right|\cdot\frac{\partial\left(f_\theta(x(i))-y(i)\right)^2}{\partial\theta}\right] \tag{8}$$
$$=\frac{n}{\sum_{j=1}^{n}\left|f_\theta(x_j)-y_j\right|}\cdot\mathop{\mathbb{E}}_{(x,y)\sim uniform(\mathcal{T})}\left[\frac{1}{3}\cdot\frac{\partial\left|f_\theta(x)-y\right|^3}{\partial\theta}\right]. \tag{9}$$

Setting $c=\frac{\sum_{i=1}^{n}|f_\theta(x_i)-y_i|}{n}$ completes the proof. □

1.4 PROOF FOR THEOREM 2

Theorem 2. (Fast early learning, detailed version) Let n be a positive integer (i.e., the number of training samples). Let $x_t,\tilde{x}_t\in\mathbb{R}^n$ be the target estimates of all samples at time $t, t\geq 0$, and $x(i)(i\in[n],[n]\overset{\text{def}}{=}\{1,2,...,n\})$ be the ith element in the vector. We define the objectives:

$$\ell_2(x,y)\overset{\text{def}}{=}\frac{1}{2}\sum_{i=1}^{n}\left(x(i)-y(i)\right)^2,\quad \ell_3(x,y)\overset{\text{def}}{=}\frac{1}{3}\sum_{i=1}^{n}\left|x(i)-y(i)\right|^3.$$

Let $\{x_t\}_{t\geq 0}$ and $\{\tilde{x}_t\}_{t\geq 0}$ be generated by using ℓ_2,ℓ_3 objectives respectively. Then define the total absolute prediction errors respectively:

$$\delta_t\overset{\text{def}}{=}\sum_{i=1}^{n}\delta_t(i)=\sum_{i=1}^{n}\left|x_t(i)-y(i)\right|,\quad \tilde{\delta}_t\overset{\text{def}}{=}\sum_{i=1}^{n}\tilde{\delta}_t(i)=\sum_{i=1}^{n}\left|\tilde{x}_t(i)-y(i)\right|,$$

where $y(i)\in\mathbb{R}$ is the training target for the ith training sample. That is, $\forall i\in[n]$,

$$\frac{dx_t(i)}{dt}=-\eta\cdot\frac{d\ell_2(x_t,y)}{dx_t(i)},\quad \frac{d\tilde{x}_t(i)}{dt}=-\eta'\cdot\frac{d\ell_3(\tilde{x}_t,y)}{d\tilde{x}_t(i)}.$$

Assume the same initialization $x_0=\tilde{x}_0$. Then:

(i) For all $i \in [n]$, define the following hitting time, which is the minimum time that the absolute error takes to be $\leq \epsilon(i)$,

$$t_\epsilon(i) \overset{\text{def}}{=} \min_t\{t \geq 0 : \delta_t(i) \leq \epsilon(i)\}, \quad \tilde{t}_\epsilon(i) \overset{\text{def}}{=} \min_t\{t \geq 0 : \tilde{\delta}_t(i) \leq \epsilon(i)\}.$$

Then, $\forall i \in [n]$ s.t. $\delta_0(i) > \frac{\eta}{\eta'}$, given an absolute error threshold $\epsilon(i) \geq 0$, there exists $\epsilon_0(i) \in (0, \frac{\eta}{\eta'})$, such that for all $\epsilon(i) > \epsilon_0(i), t_\epsilon(i) \geq \tilde{t}_\epsilon(i)$.

(ii) Define the following quantity, for all $t \geq 0$,

$$H_t^{-1} \overset{\text{def}}{=} \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{\delta_t(i)} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{|x_t(i) - y(i)|}. \tag{10}$$

Given any $0 < \epsilon \leq \delta_0 = \sum_{i=1}^{n} \delta_0(i)$, define the following hitting time, which is the minimum time that the total absolute error takes to be $\leq \epsilon$,

$$t_\epsilon \overset{\text{def}}{=} \min_t\{t \geq 0 : \delta_t \leq \epsilon\}, \quad \tilde{t}_\epsilon \overset{\text{def}}{=} \min_t\{t \geq 0 : \tilde{\delta}_t \leq \epsilon\}. \tag{11}$$

If there exists $\delta_0 \in \mathbb{R}$ and $0 < \epsilon \leq \delta_0$ such that the following holds,

$$H_0^{-1} \leq \frac{\eta}{\eta'} \cdot \frac{\log(\delta_0/\epsilon)}{\frac{\delta_0}{\epsilon} - 1}, \tag{12}$$

then we have, $t_\epsilon \geq \tilde{t}_\epsilon$, which means gradient descent using the cubic loss function will achieve the total absolute error threshold ϵ faster than using the square loss function.

Proof. **First part. (i).** For the ℓ_2 loss function, for all $i \in [n]$ and $t \geq 0$, we have,

$$\begin{aligned}
\frac{d\delta_t(i)}{dt} &= \sum_{j=1}^{n} \frac{d\delta_t(i)}{dx_t(j)} \cdot \frac{dx_t(j)}{dt} && (13)\\
&= \frac{d\delta_t(i)}{dx_t(i)} \cdot \frac{dx_t(i)}{dt} \qquad \left(\frac{d\delta_t(i)}{dx_t(j)} = 0 \text{ for all } i \neq j\right) && (14)\\
&= \operatorname{sgn}\{x_t(i) - y(i)\} \cdot (-\eta) \cdot \frac{d\ell_2(x_t, y)}{dx_t(i)} && (15)\\
&= \operatorname{sgn}\{x_t(i) - y(i)\} \cdot (-\eta) \cdot (x_t(i) - y(i)) && (16)\\
&= -\eta\,|x_t(i) - y(i)| && (17)\\
&= -\eta \cdot \delta_t(i), && (18)
\end{aligned}$$

which implies that,

$$\frac{d\{\log \delta_t(i)\}}{dt} = \frac{1}{\delta_t(i)} \cdot \frac{d\delta_t(i)}{dt} = -\eta. \tag{19}$$

Taking integral, we have,

$$\log \delta_t(i) - \log \delta_0(i) = -\eta \cdot t. \tag{20}$$

Let $\delta_t(i) = \epsilon(i)$. We have,

$$t_\epsilon(i) \overset{\text{def}}{=} \frac{1}{\eta} \cdot \log\left(\frac{\delta_0(i)}{\delta_t(i)}\right) = \frac{1}{\eta} \cdot \log\left(\frac{\delta_0(i)}{\epsilon(i)}\right). \tag{21}$$

On the other hand, for the ℓ_3 loss function, we have,

$$
\begin{aligned}
\frac{d\{\tilde{\delta}_t(i)^{-1}\}}{dt} &= \sum_{j=1}^{n} \frac{d\tilde{\delta}_t(i)^{-1}}{d\tilde{x}_t(j)} \cdot \frac{d\tilde{x}_t(j)}{dt} && (22)\\
&= \frac{d\tilde{\delta}_t(i)^{-1}}{d\tilde{x}_t(i)} \cdot \frac{d\tilde{x}_t(i)}{dt} && (23)\\
&= -\frac{1}{\tilde{\delta}_t(i)^2} \cdot \frac{d\tilde{\delta}_t(i)}{d\tilde{x}_t(i)} \cdot \frac{d\tilde{x}_t(i)}{dt} && (24)\\
&= -\frac{1}{\left(\tilde{x}_t(i) - y(i)\right)^2} \cdot \operatorname{sgn}\{\tilde{x}_t(i) - y(i)\} \cdot (-\eta') \cdot \frac{d\ell_3(\tilde{x}_t, y)}{d\tilde{x}_t(i)} && (25)\\
&= -\frac{\operatorname{sgn}\{\tilde{x}_t(i) - y(i)\}}{\left(\tilde{x}_t(i) - y(i)\right)^2} \cdot (-\eta') \cdot \left(\tilde{x}_t(i) - y(i)\right)^2 \cdot \operatorname{sgn}\{\tilde{x}_t(i) - y(i)\} && (26)\\
&= \eta'. && (27)
\end{aligned}
$$

Taking integral, we have,

$$\frac{1}{\tilde{\delta}_t(i)} - \frac{1}{\tilde{\delta}_0(i)} = \eta' \cdot t. \tag{28}$$

Let $\tilde{\delta}_t(i) = \epsilon(i)$. We have,

$$\tilde{t}_\epsilon(i) \overset{\text{def}}{=} \frac{1}{\eta'} \cdot \left(\frac{1}{\tilde{\delta}_t(i)} - \frac{1}{\tilde{\delta}_0(i)} \right) = \frac{1}{\eta'} \cdot \left(\frac{1}{\epsilon(i)} - \frac{1}{\tilde{\delta}_0(i)} \right). \tag{29}$$

Then we have,

$$
\begin{aligned}
t_\epsilon(i) - \tilde{t}_\epsilon(i) &= \frac{1}{\eta} \cdot \log\left(\frac{\delta_0(i)}{\epsilon(i)}\right) - \frac{1}{\eta'} \cdot \left(\frac{1}{\epsilon(i)} - \frac{1}{\tilde{\delta}_0(i)} \right) && (30)\\
&= \frac{1}{\eta} \cdot \left[\left(\log\frac{1}{\epsilon(i)} - \frac{\eta}{\eta'} \cdot \frac{1}{\epsilon(i)} \right) - \left(\log\frac{1}{\delta_0(i)} - \frac{\eta}{\eta'} \cdot \frac{1}{\tilde{\delta}_0(i)} \right) \right]. && (31)
\end{aligned}
$$

According to $x_0(i) = \tilde{x}_0(i)$, we have

$$
\begin{aligned}
\delta_0(i) &= |x_t(i) - y(i)| && (32)\\
&= |\tilde{x}_t(i) - y(i)| && (33)\\
&= \tilde{\delta}_0(i). && (34)
\end{aligned}
$$

Define the following function, for all $x > 0$,

$$f(x) = \log\frac{1}{x} - \frac{\eta}{\eta'} \cdot \frac{1}{x}. \tag{35}$$

We have, the continuous function f is monotonically increasing for $x \in (0, \frac{\eta}{\eta'}]$ and monotonically decreasing for $x \in (\frac{\eta}{\eta'}, \infty)$. Also, note that, $\max_{x>0} f(x) = f(\frac{\eta}{\eta'}) = \log\frac{\eta'}{\eta} - 1$, $\lim_{x\to 0} f(x) = \lim_{x\to\infty} f(x) = -\infty$.

Given $\delta_0(i) = \tilde{\delta}_0(i) > \frac{\eta}{\eta'}$, we have $f(\delta_0(i)) < f(\frac{\eta}{\eta'}) = \log\frac{\eta'}{\eta} - 1$. According to the intermediate value theorem, there exists $\epsilon_0(i) \in (0, \frac{\eta}{\eta'})$, such that $f(\epsilon_0(i)) = f(\delta_0(i))$. Since $f(\cdot)$ is monotonically increasing on $(0, \frac{\eta}{\eta'}]$ and monotonically decreasing on $(\frac{\eta}{\eta'}, \infty)$, for all $\epsilon(i) \in [\epsilon_0(i), \delta_0(i)]$, we have $f(\epsilon(i)) \geq f(\delta_0(i))$[1]. Therefore, we have,

$$t_\epsilon(i) - \tilde{t}_\epsilon(i) = \frac{1}{\eta} \cdot \left(f(\epsilon(i)) - f(\delta_0(i))\right) \geq 0. \tag{36}$$

[1]Note that $\epsilon(i) < \delta_0(i)$ by the design of using gradient descent updating rule. If the two are equal, $t_\epsilon(i) = \tilde{t}_\epsilon(i) = 0$ holds trivially.

Second part. (ii). For the square loss function, we have, for all $t \ge 0$,

$$\frac{d\delta_t}{dt} = \sum_{i=1}^{n} \frac{d\delta_t(i)}{dt} \tag{37}$$

$$= -\eta \cdot \sum_{i=1}^{n} \delta_t(i) \qquad \text{(by eq. (13))} \tag{38}$$

$$= -\eta \cdot \delta_t, \tag{39}$$

which implies that,

$$\frac{d\{\log \delta_t\}}{dt} = \frac{1}{\delta_t} \cdot \frac{d\delta_t}{dt} = -\eta. \tag{40}$$

Taking integral, we have,

$$\log \delta_t - \log \delta_0 = -\eta \cdot t. \tag{41}$$

Let $\delta_t = \epsilon$. We have,

$$t_\epsilon \stackrel{\text{def}}{=} \frac{1}{\eta} \cdot \log\left(\frac{\delta_0}{\delta_t}\right) = \frac{1}{\eta} \cdot \log\left(\frac{\delta_0}{\epsilon}\right). \tag{42}$$

After t_ϵ time, for all $i \in [n]$, we have,

$$\delta_{t_\epsilon}(i) = \delta_0(i) \cdot \exp\{-\eta \cdot t_\epsilon\}. \tag{43}$$

On the other hand, for the cubic loss function, we have, for all $t \ge 0$,

$$\frac{dH_t^{-1}}{dt} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{d\{\tilde{\delta}_t(i)^{-1}\}}{dt} \tag{44}$$

$$= \eta'. \qquad \text{(by eq. (22))} \tag{45}$$

Taking integral, we have,

$$H_t^{-1} - H_0^{-1} = \eta' \cdot t, \tag{46}$$

which means given a H_t^{-1} value, we can calculate the hitting time as,

$$t = \frac{1}{\eta'} \cdot \left(H_t^{-1} - H_0^{-1}\right). \tag{47}$$

Now consider after t_ϵ time, using gradient descent with the square loss function we have $\delta_{t_\epsilon}(i) = \delta_0(i) \cdot \exp\{-\eta \cdot t_\epsilon\}$ for all $i \in [n]$, which corresponds to,

$$H_{t_\epsilon}^{-1} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{\delta_{t_\epsilon}(i)} \tag{48}$$

$$= \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{\delta_0(i) \cdot \exp\{-\eta \cdot t_\epsilon\}}. \qquad \text{(by eq. (43))} \tag{49}$$

Therefore, the hitting time of using gradient descent with the cubic loss function to achieve the $H_{t_\epsilon}^{-1}$ value is,

$$\begin{aligned}
\tilde{t}_\epsilon &= \frac{1}{\eta'} \cdot \left(H_{t_\epsilon}^{-1} - H_0^{-1}\right) && (50)\\
&= \frac{1}{\eta'} \cdot \left(\frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{\delta_0(i) \cdot \exp\{-\eta \cdot t_\epsilon\}} - \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{\delta_0(i)}\right) && (51)\\
&= \frac{1}{\eta'} \cdot \left(\exp\{\eta \cdot t_\epsilon\} - 1\right) \cdot H_0^{-1} && (52)\\
&\leq \frac{1}{\eta} \cdot \left(\exp\{\eta \cdot t_\epsilon\} - 1\right) \cdot \frac{\log\left(\delta_0/\epsilon\right)}{\frac{\delta_0}{\epsilon} - 1} \qquad \text{(by eq. (12))} && (53)\\
&= \frac{1}{\eta} \cdot \left(\frac{\delta_0}{\epsilon} - 1\right) \cdot \frac{\log\left(\delta_0/\epsilon\right)}{\frac{\delta_0}{\epsilon} - 1} && (54)\\
&= \frac{1}{\eta} \cdot \log\left(\frac{\delta_0}{\epsilon}\right) && (55)\\
&= t_\epsilon, \qquad \text{(by eq. (42))} && (56)
\end{aligned}$$

finishing the proof. □

Remark. Figure 1 shows the function $f(x) = \ln \frac{1}{x} - \frac{1}{x}, x > 0$. Fix arbitrary $x' > 1$, there will be another root $\epsilon_0 < 1$ s.t. $f(\epsilon_0) = f(x')$. However, there is no real-valued solution for ϵ_0. The solution in $\mathbb{C}$ is $\epsilon_0 = -\frac{1}{W(\log 1/\delta_0 - 1/\delta_0 - \pi i)}$, where $W(\cdot)$ is a Wright Omega function. Hence, finding the exact value of ϵ_0 would require a definition of ordering on complex plane. Our current theorem statement is sufficient for the purpose of characterizing convergence rate. The theorem states that there always exists some desired low error level < 1, minimizing the square loss converges slower than the cubic loss.

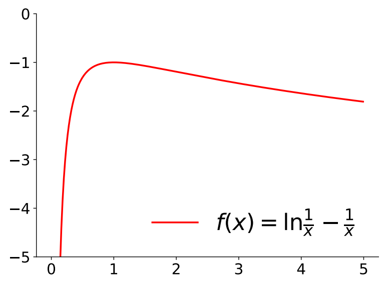
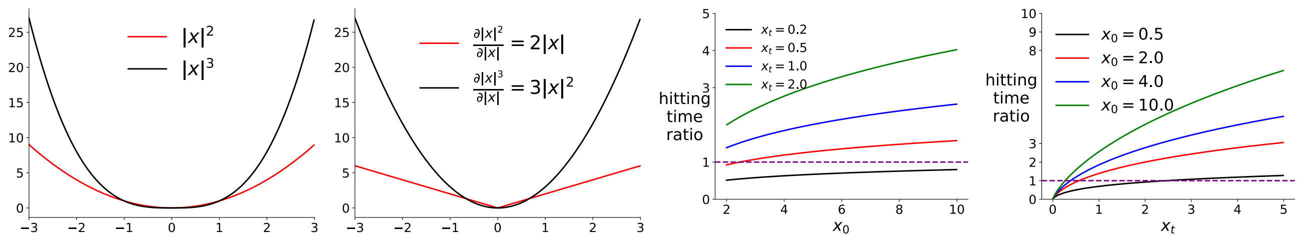


Figure 1: The function $f(x) = \ln \frac{1}{x} - \frac{1}{x}, x > 0$. The function reaches maximum at $x = 1$.

Simulations. The theorem says that if we want to minimize our loss function to certain small nonzero error level, the cubic loss function offers faster convergence rate. Intuitively, cubic loss provides sharper gradient information when the loss is large as shown in Figure 2(a)(b). Here we provides a simulation. Consider the following minimization problems: $\min_{x \geq 0} x^2$ and $\min_{x \geq 0} x^3$. For implementation and visualization convenience, we use the hitting time formulae $t_\epsilon = \frac{1}{\eta} \cdot \ln\left\{\frac{\delta_0}{\epsilon}\right\}, \tilde{t}_\epsilon = \frac{1}{\eta} \cdot \left(\frac{1}{\epsilon} - \frac{1}{\delta_0}\right)$ derived in the proof, to compute the hitting time ratio $\frac{t_\epsilon}{\tilde{t}_\epsilon}$ under different initial values x_0 and final error value ϵ. In Figure 2(c)(d), we can see that it usually takes a significantly shorter time for the cubic loss to reach a certain x_t with various initial x_0 values.

1.5 ERROR BOUND BETWEEN SAMPLING DISTRIBUTIONS

We now provide the error bound between the sampling distribution estimated by using a true model and a learned model. We denote the transition probability distribution under policy π and the true model as $\mathcal{P}^\pi(r, s'|s)$, and the learned model as $\hat{\mathcal{P}}^\pi(r, s'|s)$. Let $p(s)$ and $\hat{p}(s)$ be the convergent distributions described in the above sampling method by using the true and learned models respectively. Let $d_{tv}(\cdot, \cdot)$ be the total variation distance between the two probability distributions. Define $u(s) \overset{\text{def}}{=} |\delta(s, y(s))|, \hat{u}(s) \overset{\text{def}}{=} |\delta(s, \hat{y}(s))|, Z \overset{\text{def}}{=} \int_{s \in \mathcal{S}} u(s) ds, \hat{Z} \overset{\text{def}}{=} \int_{s \in \mathcal{S}} \hat{u}(s) ds$. Then we have the following bound.



(a) cubic v.s. square (b) |derivative| (c) x_0 v.s. hitting time (d) x_t v.s. hitting time

Figure 2: (a) show cubic v.s. square function. (b) shows their absolute derivatives. (c) shows the hitting time ratio v.s. initial value x_0 under different target value x_t. (d) shows the ratio v.s. the target x_t to reach under different x_0. Note that a ratio larger than 1 indicates a longer time to reach the given x_t for the square loss.

Theorem 1. *Assume: 1) the reward magnitude is bounded $|r| \le R_{max}$ and define $V_{max} \overset{\text{def}}{=} \frac{R_{max}}{1-\gamma}$; 2) the largest model error for a single state is $\epsilon_s \overset{\text{def}}{=} \max_s d_{tv}(\mathcal{P}^\pi(\cdot|s), \hat{\mathcal{P}}^\pi(\cdot|s))$ and the total model error is bounded, i.e. $\epsilon \overset{\text{def}}{=} \int_{s\in\mathcal{S}} \epsilon_s ds < \infty$. Then $\forall s \in \mathcal{S}, |p(s) - \hat{p}(s)| \le \min(\frac{V_{max}(p(s)\epsilon+\epsilon_s)}{\hat{Z}}, \frac{V_{max}(\hat{p}(s)\epsilon+\epsilon_s)}{Z})$.*

Proof. First, we bound the estimated temporal difference error. Fix an arbitrary state $s \in \mathcal{S}$, it is sufficient the consider the case $u(s) > \hat{u}(s)$, then

$$
\begin{aligned}
&|u(s) - \hat{u}(s)| = u(s) - \hat{u}(s) \\
=&\mathbb{E}_{(r,s')\sim\mathcal{P}^\pi}[r + \gamma v^\pi(s')] - \mathbb{E}_{(r,s')\sim\hat{\mathcal{P}}^\pi}[r + \gamma v^\pi(s')] \\
=&\int_{s,r} (r + \gamma v^\pi(s))(\mathcal{P}^\pi(s', r|s) - \hat{\mathcal{P}}^\pi(s', r|s))ds'dr \\
\le&(R_{max} + \gamma\frac{R_{max}}{1-\gamma})\int_{s,r}(\mathcal{P}^\pi(s', r|s) - \hat{\mathcal{P}}^\pi(s', r|s))ds'dr \\
\le&V_{max}d_{tv}(\mathcal{P}^\pi(\cdot|s), \hat{\mathcal{P}}^\pi(\cdot|s)) \le V_{max}\epsilon_s
\end{aligned}
$$

Now, we show that $|Z - \hat{Z}| \le V_{max}\epsilon$.

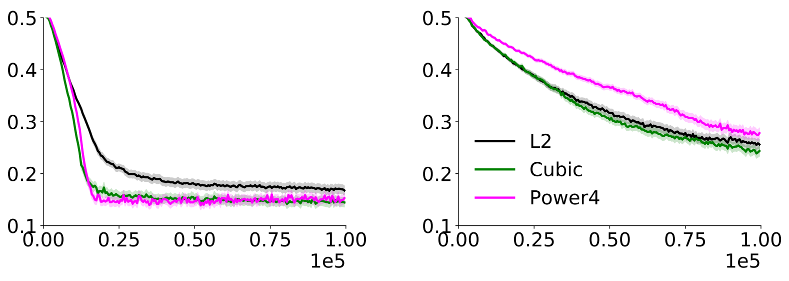
$$
\begin{aligned}
|Z - \hat{Z}| &= |\int_{s\in\mathcal{S}} u(s)ds - \int_{s\in\mathcal{S}} \hat{u}(s)ds| = |\int_{s\in\mathcal{S}}(u(s) - \hat{u}(s))ds| \\
&\le \int_{s\in\mathcal{S}} |u(s) - \hat{u}(s)|ds \le V_{max}\int_{s\in\mathcal{S}} \epsilon_s ds = V_{max}\epsilon
\end{aligned}
$$

Consider the case $p(s) > \hat{p}(s)$ first.

$$
\begin{aligned}
p(s) - \hat{p}(s) &= \frac{u(s)}{Z} - \frac{\hat{u}(s)}{\hat{Z}} \\
&\le \frac{u(s)}{Z} - \frac{u(s) - V_{max}\epsilon_s}{\hat{Z}} = \frac{u(s)\hat{Z} - u(s)Z + ZV_{max}\epsilon_s}{Z\hat{Z}} \\
&\le \frac{u(s)V_{max}\epsilon + ZV_{max}\epsilon_s}{Z\hat{Z}} = \frac{V_{max}(p(s)\epsilon + \epsilon_s)}{\hat{Z}}
\end{aligned}
$$

Meanwhile, below inequality should also hold:

$$
\begin{aligned}
p(s) - \hat{p}(s) &= \frac{u(s)}{Z} - \frac{\hat{u}(s)}{\hat{Z}} \le \frac{\hat{u}(s) + V_{max}\epsilon_s}{Z} - \frac{\hat{u}(s)}{\hat{Z}} \\
&= \frac{\hat{u}(s)\hat{Z} - \hat{u}(s)Z + \hat{Z}V_{max}\epsilon_s}{Z\hat{Z}} \le \frac{V_{max}(\hat{p}(s)\epsilon + \epsilon_s)}{Z}
\end{aligned}
$$



(a) $\sigma = 0.1$ (b) $\sigma = 0.5$

Figure 3: Figure(a)(b) show the testing RMSE as a function of number of mini-batch updates with increasing noise standard deviation σ added to the training targets. We compare the performances of **Power4(magenta)**, **L2 (black)**, **Cubic (forest green)**. The results are averaged over 50 random seeds. The shade indicates standard error. Note that the testing set is not noise-contaminated.

Because both the two inequalities must hold, when $p(s) - \hat{p}(s) > 0$, we have:

$$p(s) - \hat{p}(s) \leq \min(\frac{V_{max}(p(s)\epsilon + \epsilon_s)}{\hat{Z}}, \frac{V_{max}(\hat{p}(s)\epsilon + \epsilon_s)}{Z})$$

It turns out that the bound is the same when $p(s) \leq \hat{p}(s)$. This completes the proof. □

1.6 HIGH POWER LOSS FUNCTIONS

We would like to point out that directly using a high power objective in general problems is unlikely to have an advantage.

First, notice that our convergence rate is characterized w.r.t. to the expected updating rule, not stochastic gradient updating rule. When using a stochastic sample to estimate the gradient, high power objectives are sensitive to the outliers as they augment the effect of noise. Robustness to outliers is also the motivation behind the Huber loss [Huber, 1964] which, in fact, uses low power error in most places so it can be less sensitive to outliers.

We conduct experiments to examine the effect of noise on using high power objectives. We use the same dataset as described in Section 3.3. We use a training set with 4k training examples. The naming rules are as follows. **Cubic** is minimizing the cubic objective (i.e. $\min_\theta \frac{1}{n}\sum_{i=1}^n |f_\theta(x_i) - y_i|^3$) by uniformly sampling, and **Power4** is $\min_\theta \frac{1}{n}\sum_{i=1}^n (f_\theta(x_i) - y_i)^4$ by uniformly sampling.

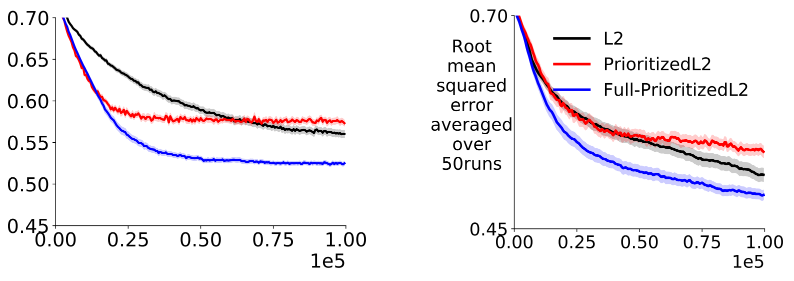
Figure 3 (a)(b) shows the learning curves of uniformly sampling for Cubic and for Power4 trained by adding noises with standard deviation $\sigma = 0.1, 0.5$ respectively to the training targets. It is not surprising that all algorithms learn slower when we increase the noise variance added to the target variables. However, one can see that *high power objectives is more sensitive to noise variance added to the targets than the regular L2*: when $\sigma = 0.1$, the higher power objectives perform better than the regular L2; after increasing σ to 0.5, Cubic becomes almost the same as L2, while Power4 becomes worse than L2.

Second, it should be noted that in our theorem, we do not characterize the convergence rate to the minimum; instead, we show the convergence rate to a certain low error solution, corresponding to early learning performance. In optimization literature, it is known that cubic power would converge slower to the minimizer as it has a relatively flat bottom. However, it may be an interesting future direction to study how to combine objectives with different powers so that optimizing the hybrid objective leads to a faster convergence rate to the optimum and is robust to outliers.

1.7 ADDITIONAL EXPERIMENTS

In this section, we include the following additional experimental results:

1. As a supplementary to Figure 1 from Section 3.3, we show the learning performance measured by training errors to show the negative effects of the two limitations.



(a) $|\mathcal{T}| = 4000$ (b) $|\mathcal{T}| = 400$

Figure 4: Figure (a)(b) show the training RMSE as a function of number of mini-batch updates with a training set containing 4k examples and another containing 400 examples respectively. We compare the performances of **Full-PrioritizedL2 (blue)**, **L2 (black)**, and **PrioritizedL2 (red)**. The results are averaged over 50 random seeds. The shade indicates standard error.

2. Empirical verification of Theorem 1 (prioritized sampling and uniform sampling on cubic power equivalence).
3. Additional results on discrete domains 1.7.3.
4. Results on an autonomous driving application 1.7.4.
5. Results on MazeGridWorld from Pan et al. [2020].

1.7.1 Training Error Corresponding to Figure 1 from Section 3.3

Note that our Theorem 1 and 2 characterize the expected gradient calculated on the training set; hence it is sufficient to examine the learning performances measured by training errors. However, the testing error is usually the primary concern, so we put the testing error in the main body. As a sanity check, we also investigate the learning performances measured by training error and find that those algorithms behave similarly as shown in Figure 4 where the algorithms are trained by using training sets with decreasing training examples from (a) to (b). As we reduce the training set size, Full-PrioritizedL2 is closer to L2. Furthermore, PrioritizedL2 is always worse than Full-PrioritizedL2. These observations show the negative effects resulting from the issues of outdated priorities and insufficient sample space coverage.

1.7.2 Empirical verification of Theorem 1

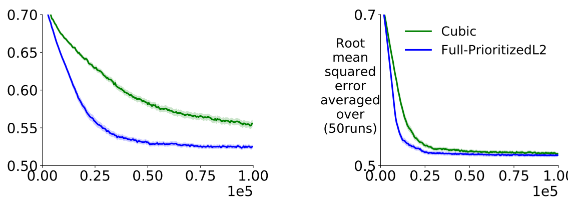
Theorem 1 states that the expected gradient of doing prioritized sampling on mean squared error is equal to the gradient of doing uniformly sampling on cubic power loss. As a result, we expect that the learning performance on the training set (note that we calculate gradient by using training examples) should be similar when we use a large mini-batch update as the estimate of the expectation terms become close.

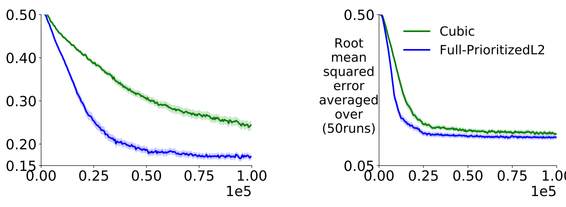
We use the same dataset as described in Section 3.3 and keep using training size 4k. Figure 5(a)(b) shows that when we increase the mini-batch size, *the two algorithms Full-PrioritizedL2 and Cubic are becoming very close to each other*, verifying our theorem.

Note that our theorem characterizes the expected gradient calculated on the training set; hence it is sufficient to examine the learning performances measured by training errors. However, usually, the testing error is the primary concern. For completeness, we also investigate the learning performances measured by testing error and find that the tested algorithms behave similarly as shown in Figure 5(c)(d).

1.7.3 Additional Results on Discrete Benchmark Domains

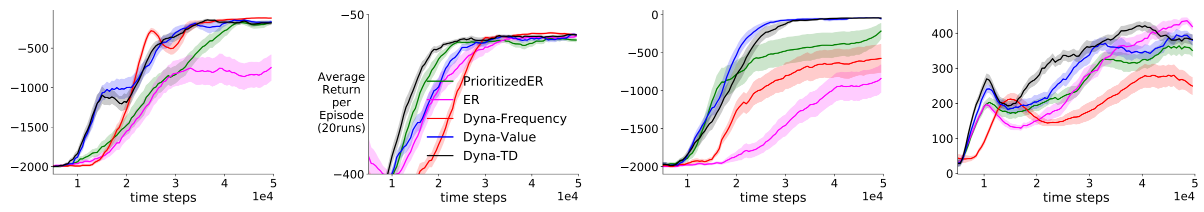
Figure 6 shows the empirical results of our algorithm on the discrete domains with plan steps $= 5$.





(a) b=128, $\sigma = 0.5$ (b) b=512, $\sigma = 0.5$ (c) b=128, $\sigma = 0.5$ (d) b=512, $\sigma = 0.5$

Figure 5: Figure(a)(b) show the training RMSE as a function of number of mini-batch updates with increasing mini-batch size b. Figure (c)(d) show the testing RMSE. We compare the performances of **Full-PrioritizedL2 (blue)**, **Cubic (forest green)**. As we increase the mini-batch size, the two performs more similar to each other. The results are averaged over 50 random seeds. The shade indicates standard error.

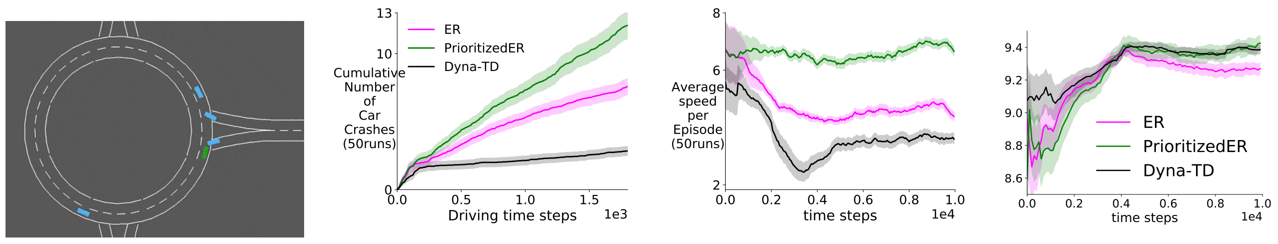


(a) MountainCar (b) Acrobot (c) GridWorld (d) CartPole

Figure 6: Episodic return v.s. environment time steps. We show evaluation learning curves of **Dyna-TD (black)**, **Dyna-Frequency (red)**, **Dyna-Value (blue)**, **PrioritizedER (forest green)**, and **ER(magenta)** with planning updates $n = 5$.

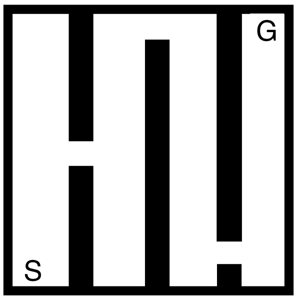
1.7.4 Autonomous Driving Application

We study the practical utility of our method in a relatively large autonomous driving application [Leurent, 2018] with an online learned model. We use the roundabout-v0 domain (Figure 7 (a)). The agent learns to go through a roundabout by lane change and longitude control. The reward is designed such that the car should go through the roundabout as fast as possible without collision. We observe that all algorithms perform similarly when evaluating algorithms by episodic return (Figure 7 (d)). In contrast, there is a significantly lower number of car crashes with the policy learned by our algorithm, as shown in Figure 7(b). Figure 7 (c) suggests that ER and PrioritizedER gain reward mainly due to fast speed which potentially incur more car crashes. The conventional prioritized ER method still incurs many crashes, which may indicate its prioritized sampling distribution does not provide enough crash experiences to learn.

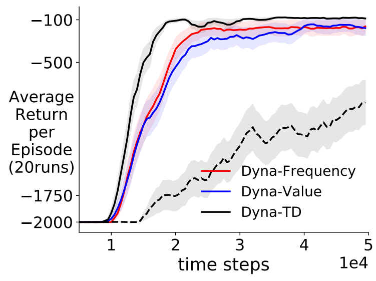


(a) roundabout (b) Num of car crashes (c) Avg. speed (d) Episodic return

Figure 7: (a) shows the roundabout domain with $\mathcal{S} \subset \mathbb{R}^{90}$. (b) shows crashes v.s. total driving time steps during policy evaluation. (c) shows the average speed per evaluation episode v.s. environment time steps. (d) shows the episodic return v.s. trained environment time steps. We show **Dyna-TD (black)** with an online learned model, **PrioritizedER (forest green)**, and **ER (magenta)**. Results are averaged over 50 random seeds after smoothing over a window of size 30. The shade indicates standard error.



(a) MazeGridWorld



(b) MazeGW, $n = 30$

Figure 8: Figure(a) shows MazeGridWorld(GW) taken from Pan et al. [2020] and the learning curves are in (b). We show evaluation learning curves of **Dyna-TD (black)**, **Dyna-Frequency (red)**, and **Dyna-Value (blue)**. The dashed line indicates Dyna-TD trained with an online learned model. All results are averaged over 20 random seeds after smoothing over a window of size 30. The shade indicates standard error.

1.7.5 Results on MazeGridWorld Domain

In Figure 8, we demonstrate that our algorithm can work better than Dyna-Frequency on a MazeGridWorld domain Pan et al. [2020], where Dyna-Frequency was shown to be superior to Dyna-Value and model-free baselines. This result further confirms the usefulness of our sampling approach.

1.8 REPRODUCIBLE RESEARCH

Our implementations are based on tensorflow with version 1.13.0 Abadi et al. [2015]. We use Adam optimizer Kingma and Ba [2014] for all experiments. The code is available at `https://github.com/yannickycpan/reproduceRL.git`.

1.8.1 Reproduce experiments before Section 5

Supervised learning experiment. For the supervised learning experiment shown in section 3, we use 32×32 tanh units neural network, with learning rate swept from $\{0.01, 0.001, 0.0001, 0.00001\}$ for all algorithms. We compute the constant c as specified in the Theorem 1 at each time step for Cubic loss. We compute the testing error every 500 iterations/mini-batch updates and our evaluation learning curves are plotted by averaging 50 random seeds. For each random seed, we randomly split the dataset to testing set and training set and the testing set has 1k data points. Note that the testing set is not noise-contaminated.

Reinforcement Learning experiments in Section 3. We use a particularly small neural network 16×16 to highlight the issue of incomplete priority updating. Intuitively, a large neural network may be able to memorize each state's value and thus updating one state's value is less likely to affect others. We choose a small neural network, in which case a complete priority updating for all states should be very important. We set the maximum ER buffer size as 10k and mini-batch size as 32. The learning rate is chosen from $\{0.0001, 0.001\}$ and the target network is updated every 1k steps.

Distribution distance computation in Section 4. We now introduce the implementation details for Figure 3. The distance is estimated by the following steps. First, in order to compute the desired sampling distribution, we discretize the domain into 50×50 grids and calculate the absolute TD error of each grid (represented by the left bottom vertex coordinates) by using the true environment model and the current learned Q function. We then normalize these priorities to get probability distribution p^*. Note that this distribution is considered as the desired one since we have access to all states across the state space with priorities computed by current Q-function at each time step. Second, we estimate our sampling distribution by randomly sampling 3k states from search-control queue and count the number of states falling into each discretized grid and normalize these counts to get p_1. Third, for comparison, we estimate the sampling distribution of the conventional prioritized ER Schaul et al. [2016] by sampling 3k states from the prioritized ER buffer and count the states falling into each grid and compute its corresponding distribution p_2 by normalizing the counts. Then we compute the distances of p_1, p_2 to p^* by two weighting schemes: 1) on-policy weighting: $\sum_{j=1}^{2500} d^\pi(s_j)|p_i(s_j) - p^*(s_j)|, i \in \{1, 2\}$, where d^π is approximated by uniformly sample 3k states from a recency buffer and normalizing their visitation counts on the discretized GridWorld; 2) uniform weighting: $\frac{1}{2500}\sum_{j=1}^{2500}|p_i(s_j)-p^*(s_j)|, i \in \{1,2\}$. We examine the two weighting schemes because of two considerations: for the on-policy weighting, we concern about the asymptotic convergent behavior and want to down-weight those states with relatively high TD error but get rarely visited as the policy gets close to optimal; uniform weighting makes more sense during early learning stage, where we consider all states are equally important and want the agents to sufficiently explore the whole state space.

Computational cost v.s. performance in Section 4. The setting is the same as we used for Section 5. We use plan step/updates=10 to generate that learning curve.

1.8.2 Reproduce experiments in Section 5

For our algorithm, the pseudo-code with concrete parameter settings is presented in Algorithm 3.

Common settings. For all discrete control domains other than roundabout-v0, we use 32×32 neural network with ReLu hidden units except the Dyna-Frequency which uses tanh units as suggested by the author Pan et al. [2020]. This is one of its disadvantages: the search-control of Dyna-Frequency requires the computation of Hessian-gradient product and it is empirically observed that the Hessian is frequently zero when using ReLu as hidden units. Except the output layer parameters which were initialized from a uniform distribution $[-0.003, 0.003]$, all other parameters are initialized using Xavier initialization Glorot and Bengio [2010]. We use mini-batch size $b = 32$ and maximum ER buffer size 50k. All algorithms use target network moving frequency 1000 and we sweep learning rate from $\{0.001, 0.0001\}$. We use warm up steps $= 5000$ (i.e. random action is taken in the first 5k time steps) to populate the ER buffer before learning starts. We keep exploration noise as 0.1 without decaying.

Hyper-parameter settings. Across RL experiments including both discrete and continuous control tasks, we are able to fix the same parameters for our hill climbing updating rule 3 $s \leftarrow s + \alpha_h \nabla_s \log|\hat{y}(s) - \max_a Q(s,a;\theta_t)| + X$, where we fix $\alpha_h = 0.1, X \sim N(0, 0.01)$.

For our algorithm Dyna-TD, we are able to keep the same parameter setting across all discrete domains: $c = 20$ and learning rate 0.001. For all Dyna variants, we fetch the same number of states ($m = 20$) from hill climbing (i.e. search-control process) as Dyna-TD does, and use $\epsilon_{accept} = 0.1$ and set the maximum number of gradient step as $k = 100$ unless otherwise specified.

Our Prioritized ER is implemented as the proportional version with sum tree data structure. To ensure fair comparison, since all model-based methods are using mixed mini-batch of samples, we use prioritized ER without importance ratio but half of mini-batch samples are uniformly sampled from the ER buffer as a strategy for bias correction. For Dyna-Value and Dyna-Frequency, we use the setting as described by the original papers.

For the purpose of learning an environment model on those discrete control domains, we use a 64×64 ReLu units neural network to predict $s' - s$ and reward given a state-action pair s, a; and we use mini-batch size 128 and learning rate 0.0001 to minimize the mean squared error objective for training the environment model.

Environment-specific settings. All of the environments are from OpenAI [Brockman et al., 2016] except that: 1) the GridWorld envirnoment is taken from Pan et al. [2019] and the MazeGridWorld is from Pan et al. [2020]; 2) Roundabout-v0 is from Leurent et al. [2019]. For all OpenAI environments, we use the default setting except on Mountain Car where we set the episodic length limit to 2k. The GridWorld has state space $\mathcal{S} = [0,1]^2$ and each episode starts from the left bottom and the goal area is at the top right $[0.95, 1.0]^2$. There is a wall in the middle with a hole to allow the agent to pass. MazeGridWorld is a more complicated version where the state and action spaces are the same as GridWorld, but there are two walls in the middle and it takes a long time for model-free methods to be successful. On the this domain, we use the same setting as the original paper for all Dyna variants. We use exactly the same setting as described above except that we change the $Q-$ network size to 64×64 ReLu units, and number of search-control samples is $m = 50$ as used by the original paper. We refer readers to the original paper Pan et al. [2020] for more details.

On roundabout-v0 domain, we use 64×64 ReLu units for all algorithms and set mini-batch size as 64. The environment model is learned by using a 200×200 ReLu neural network trained by the same way mentioned above. For Dyna-TD, we start using the model after 5k steps and set $m = 100, k = 500$ and we do search-control every 50 environment time steps to reduce computational cost. To alleviate the effect of model error, we use only 16 out of 64 samples from the search-control queue in a mini-batch.

Algorithm 2 Dyna-TD

```
Input: $m$: number of states to fetch through search-control; $B_{sc}$: empty search-control queue; $B_{er}$: ER buffer; $\epsilon_{accept}$: threshold for accepting a state; initialize $Q$-network $Q_\theta$
for $t = 1, 2, \ldots$ do
    Observe $(s_t, a_t, s_{t+1}, r_{t+1})$ and add it to $B_{er}$
    // Hill climbing on absolute TD error
    Sample $s$ from $B_{er}$, $c \leftarrow 0, \tilde{s} \leftarrow s$
    while $c < m$ do
        $\hat{y} \leftarrow \mathbb{E}_{s',r\sim\hat{\mathcal{P}}(\cdot|s,a)}[r + \gamma \max_a Q_\theta(s', a)]$
        Update $s$ by rule (3)
        if $s$ is out of the state space then
            Sample $s$ from $B_{er}$, $\tilde{s} \leftarrow s$ // restart
            continue
        if $||\tilde{s} - s||_2/\sqrt{d} \geq \epsilon_{accept}$ then
            // $d$ is the number of state variables, i.e. $\mathcal{S} \subset \mathbb{R}^d$
            Add $s$ into $B_{sc}$, $\tilde{s} \leftarrow s$, $c \leftarrow c + 1$
    //$n$ planning updates
    for $n$ times do
        Sample a mixed mini-batch with half samples from $B_{sc}$ and half from $B_{er}$
        Update $Q$-network parameters by using the mixed mini-batch
```

On Mujoco domains Hopper and Walker2d, we use 200×100 ReLu units for all algorithms and set mini-batch size as 64. The environment model is learned by using a 200×200 ReLu neural network trained by the same way mentioned above. For Dyna-TD, we start using the model after 10k steps and set $m = 100, k = 500$ and we do search-control every 50 environment time steps to reduce computational cost. To alleviate the effect of model error, we use only 16 out of 64 samples from the search-control queue in a mini-batch.

Algorithm 3 Dyna-TD with implementation details

```
**Input or notations:** $k = 20$: number search-control states to acquire by hill climbing, $k_b = 100$: the budget of maximum number of hill climbing steps; $\rho = 0.5$: percentage of samples from search-control queue, $d : \mathcal{S} \subset \mathbb{R}^d$; empty search-control queue $B_{sc}$ and ER buffer $B_{er}$
empirical covariance matrix: $\hat{\Sigma}_s \leftarrow \mathbf{I}$
$\mu_{ss} \leftarrow \mathbf{0} \in \mathbb{R}^{d\times d}, \mu_s \leftarrow \mathbf{0} \in \mathbb{R}^d$ (auxiliary variables for computing empirical covariance matrix, sample average will be maintained for $\mu_{ss}, \mu_s$)
$n_\tau \leftarrow 0$: count for parameter updating times, $\tau \leftarrow 1000$ target network updating frequency
$\epsilon_{accept} \leftarrow 0$: threshold for accepting a state
Initialize $Q$ network $Q_\theta$ and target $Q$ network $Q_{\theta'}$
**for** $t = 1, 2, \dots$ **do**
    Observe $(s, a, s', r)$ and add it to $B_{er}$
    $\mu_{ss} \leftarrow \frac{\mu_{ss}(t-1)+ss^\top}{t}, \mu_s \leftarrow \frac{\mu_s(t-1)+s}{t}$
    $\hat{\Sigma}_s \leftarrow \mu_{ss} - \mu_s\mu_s^\top$
    $\epsilon_{accept} \leftarrow (1-\beta)\epsilon_{accept} + \beta||s' - s||_2$ for $\beta = 0.001$
    // Hill climbing on absolute TD error
    Sample $s$ from $B_{er}$, $c \leftarrow 0, \tilde{s} \leftarrow s, i \leftarrow 0$
    **while** $c < k$ and $i < k_b$ **do**
        // since environment is deterministic, the environment model becomes a Dirac-delta distribution and we denote it as a deterministic function $\mathcal{M} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S} \times \mathbb{R}$
        $s', r \leftarrow \mathcal{M}(s, a)$
        $\hat{y} \leftarrow r + \gamma \max_a Q_\theta(s', a)$
        // add a smooth constant $10^{-5}$ inside the logarithm
        $s \leftarrow s + \alpha_h \nabla_s \log(|\hat{y} - \max_a Q(s, a; \theta_t)| + 10^{-5}) + X, X \sim N(0, 0.01\hat{\Sigma}_s)$
        **if** $s$ is out of the state space **then**
            // restart hill climbing
            Sample $s$ from $B_{er}$, $\tilde{s} \leftarrow s$
            **continue**
        **if** $||\tilde{s} - s||_2/\sqrt{d} \geq \epsilon_{accept}$ **then**
            Add $s$ into $B_{sc}$, $\tilde{s} \leftarrow s$, $c \leftarrow c + 1$
        $i \leftarrow i + 1$
    **for** $n$ times **do**
        Sample a mixed mini-batch $b$, with proportion $\rho$ from $B_{sc}$ and $1-\rho$ from $B_{er}$
        Update parameters $\theta$ (i.e. DQN update) with $b$
        $n_\tau \leftarrow n_\tau + 1$
        **if** $mod(n_\tau, \tau) == 0$ **then**
            $Q_{\theta'} \leftarrow Q_\theta$
```